

TN-61-TM

176377

p. 23

Maintenance Strategies for Design Recovery and Reengineering

Volume 2
FORTRAN Standards

By: Dennis Braley

(NASA-TM-108261) MAINTENANCE
STRATEGIES FOR DESIGN RECOVERY AND
REENGINEERING. VOLUME 2: FORTRAN
STANDARDS (NASA) 23 p

N93-72513

Unclass

29/61 0176377

June 1990
Review Copy

Software Technology Branch
Johnson Space Center
National Aeronautics and Space Administration
Houston, TX 77058

6/29/90

Revision 2

ACKNOWLEDGEMENTS

The authors wish to thank participants from The MITRE Corporation who helped in the preparation of this document: Lois Morgan, who merged and edited several unpublished documents to form this document, Debra McGrath, who reviewed all volumes and continues to contribute to the design of the environment, and Dona Erb, who also reviewed all volumes.

CONTENTS

1.0	Introduction	1
2.0	Need for FORTRAN Standards at JSC	2
3.0	ANSI Standards for FORTRAN 77 and FORTRAN 8X	3
3.1	Limitations of FORTRAN 77 Standards	4
3.2	Highlights of Draft FORTRAN 8X Standards	4
4.0	Proposed Local Standards to Aid Maintenance	5
4.1	Documentation Standards	5
4.2	Longer Variable Names	7
4.3	Modern Control Flow Structures	7
4.4	Grouping Subprograms into Virtual Packages	7
4.5	Data Structuring	9
4.5.1	Inter-Subprogram Communication via Calling Parameters	9
4.5.2	Inter-Subprogram Communication via COMMON Blocks	10
4.6	Input/Output	11
5.0	Summary	11
6.0	Appendix	13
6.1	Standardized Comment Statements (CDs)	13
6.2	Tool Set	14
7.0	Glossary	17
8.0	References	19

FIGURES

Figure 1	Structure of a COMGEN-Compatible Subprogram	6
Figure 2	Comparison of Ada Package and FORTRAN Virtual Package	8

TABLES

Table 1	Standards Summary	12
---------	-------------------	----

1.0 INTRODUCTION

Programs in use today generally have all of the functional and information processing capabilities required to do their specified job. However, older programs usually use obsolete technology, are not integrated properly with other programs, and are difficult to maintain. Reengineering is becoming a prominent discipline as organizations try to move their programs to more modern and maintainable technologies. Johnson Space Center's (JSC) Software Technology Branch (STB) is researching and developing a system to support reengineering older FORTRAN programs into a more maintainable form that can also be more readily translated to a modern language such as FORTRAN 8x, Ada, or C. This activity has led to the development of maintenance strategies for design recovery and reengineering. These strategies include a set of standards, a methodology, and the concepts for a software environment to support design recovery and reengineering.

These products and concepts are documented in a five volume report, *Maintenance Strategies for Design Recovery and Reengineering*, of which this is the second volume. This volume defines new proposed FORTRAN coding standards. Volume 1, *Executive Summary and Problem Statement*, contains a statement of the problem and an overview of the STB's approach toward providing an economic solution to the problem; Volume 3, *Methods*, describes the methodology based on experience gained in reengineering a large FORTRAN program; Volume 4, *Concepts for an Environment*, presents the concepts and architecture for a proposed integrated environment to support the standards and methodology. Volume 5, *A Method for Conversion of FORTRAN Programs*, records lessons learned in a pilot project that converted a large FORTRAN program to Ada.

This standards document is intended to provide programmers/analysts in the JSC community with a description of new FORTRAN coding standards proposed by the STB for use in programming in FORTRAN 77 and for upgrading existing FORTRAN code to improve its maintainability. These standards augment existing standards in the mission planning and analysis domain at JSC. Software conforming to these standards will be in a form that can be more easily maintained and/or translated into the soon-to-be-released FORTRAN 8x or other modern computer languages.

Section 2 of this volume is a statement of the need at JSC for new standards with respect to the dynamic nature of the computer industry and the difficulties encountered in maintenance and translations. Section 3 discusses the ANSI FORTRAN standards, both the limitations of the current FORTRAN 77 language standards and the highlights of the draft FORTRAN 8x standards. Section 4 defines the STB's proposed coding standards. Section 5 summarizes the new standards.

It is assumed that the reader is already familiar with the existing standards defined in the mission planning and analysis domain at JSC and documented in *Computer Program Development and Maintenance Techniques* (NASA IN 80-FM-55), *Automated Software Documentation Techniques* (NASA), and *Software Development and Maintenance Aids Catalog* (NASA IN 86-FM-27). For the reader unfamiliar with these standards and the related terminology, an appendix has been provided that includes a brief explanation of each of the "CD" comment statements that support the existing documentation standards. This appendix also provides a brief description of STB tools that support the standards. Acronyms and terms

used in this volume are defined in a glossary, and references for the document are provided in a list of references.

2.0 NEED FOR FORTRAN STANDARDS AT JSC

Based on trends in the computer industry over the last few years, it is clear that computer hardware, languages, and procedures are not static. JSC has a large installed base of FORTRAN programs; many of these programs are large, complex, and difficult to understand, resulting in maintenance problems. Many already have been converted from the original dialect to FORTRAN 77. It is necessary to consider the question, where will that code have to be in five or ten years? Three possibilities come to mind:

- FORTRAN 77 is the current standard, but the next FORTRAN standard, FORTRAN 8x, is close to release. As vendors stop supporting FORTRAN 77, existing FORTRAN programs will have to be modified to conform to the new standard or be converted to another language.
- Much of the code may move to the Ada language. This will be particularly true on Space Station Freedom work.
- With C being the language of Unix and one of the languages selected for Mission Operations Directorate (MOD) projects, such as the Flight Analysis and Design System (FADS), some of the code might be converted to the C language.

Some might ask, why worry about all of this now? We can use a translator when the time comes that we are forced to move our code forward. Although this would be a nice solution, the truth is that code translators have proven unsuccessful due to the following major reasons:

- Poor existing control flow is translated into poor control flow.
- Poor existing data structures remain poor data structures.
- Input/output translation usually produces hard to read "unnatural" code in the new language.
- Translation does not take advantage of the code and data packaging techniques available in the newer languages.

Therefore, JSC needs standards to prepare for the future conversion of FORTRAN 77 code to FORTRAN 8x, Ada, or C. It is emphasized that the standards that the STB is proposing in this document will allow FORTRAN 77 code to be reengineered and converted easily to *any* of these languages; the only difference will be the generation of the actual code in the specific target language.

The standards are also needed to support the upgrading of existing programs to incorporate newer software engineering methods that have been proven to result in better quality, more maintainable code. The implementation of these standards will make existing code more understandable to programmers/analysts who are required to take over the maintenance of

programs that they did not develop. When trying to understand a large, complex program, a person can at first feel overwhelmed by the size and complexity of the program.

3.0 ANSI STANDARDS FOR FORTRAN 77 AND FORTRAN 8X

To provide for software portability among multiple vendor workstations, JSC has brought local code in line with the ANSI standards for FORTRAN 77. However, this standard is not the end of the story for FORTRAN. Consider the following points:

- Some limitations in FORTRAN 77 are in conflict with the some of the goals of software engineering that support the maintainability of software.
- An update to the ANSI standards for FORTRAN (i.e., FORTRAN 8x) is in progress within the ANSI standards committee.
- It may be desired to convert existing FORTRAN code to another language at some time in the future.

For these reasons, this document presents coding standards that do not comply with current ANSI FORTRAN 77 standards, but are extensions needed to promote software engineering principles that improve the maintainability of existing software. These proposed coding standards are supported by FORTRAN compilers today as non-standard extensions to the language. These coding standards are recognized as a temporary trade-off of portability for maintainability until the adoption of the draft FORTRAN 8x standards. At that time, code that meets these coding standards will once again meet the ANSI FORTRAN language standards. If portability becomes an issue before then for some program with code that has been modified to meet these new coding standards, tools will be provided to automatically remove the modifications, returning the code to compliance with ANSI FORTRAN standards. In summary, only those non-standard features that meet the following criteria are recommended in this document:

- Increase significantly the quality of code.
- Can be automatically removed with tools, returning the code to ANSI FORTRAN 77 standards.
- Are supported by most current FORTRAN compilers.
- Are proposed as part of the FORTRAN 8x standard.

In the following paragraphs, some of the limitations of FORTRAN 77 are described. Then some of the highlights of the FORTRAN 8x standards are introduced. This should show that the proposed JSC coding standards will become ANSI language standards when the next FORTRAN standard is formally agreed upon.

3.1 Limitations of FORTRAN 77

The following list contains the major limitations of FORTRAN 77 that are addressed by the standards presented in this document:

- Limit of six characters in variable names.
- Lack of structured programming constructs, such as a WHILE loop.
- Lack of subprogram packaging methods similar to the constructs available in Ada or C.
- Lack of flexibility in data packaging, such as the record structures provided in most modern languages.

3.2 Highlights of Draft FORTRAN 8X Standards

Based on the available draft of FORTRAN 8x standards, the next ANSI FORTRAN standard will include the following features:

- General features.
 - Longer variable names (a maximum of 31 characters).
 - Free source format.
 - Error event handling similar to Ada exception handling.
- New control flow structures.
 - Block DO.
 - Form of the block DO that is equivalent to the DO WHILE.
 - CASE structure.
- Modularization features.
 - MODULE structure similar to the Ada package and C file features.
 - Internal subroutines.
 - Calling argument interface specifications similar to Ada specification and C prototype features.
- Data structures similar to Ada, C, and Pascal.
 - Record structure.

- User-defined data types.
- Include capability.
- Ability to turn off the FORTRAN implicit typing feature.
- Declaration of data constants.

These features will greatly increase the power of the language. They will allow existing code to migrate over time to better engineered and easier to maintain code. Only the features listed above that are currently supported by most FORTRAN compilers as extensions to the FORTRAN 77 standard are recommended as coding standards for JSC.

4.0 PROPOSED LOCAL CODING STANDARDS TO AID MAINTENANCE

This section contains coding standards that are recommended for writing new FORTRAN programs or upgrading existing programs. FORTRAN partially supports the newer software engineering concepts, such as abstractions, information hiding, modularity, localizations, and structured programming. It is possible to write FORTRAN that simulates the features that are not supported. A FORTRAN program using the guidelines presented in this volume will be easier to maintain and easier to convert to FORTRAN 8x, Ada, or C. It might even be possible to perform a semi-automatic translation rather than a conversion process to update the code.

These guidelines are presented with the understanding that no matter how stringent rules are, they can be evaded. Unless the programmers and engineers believe in the purpose and intent of a set of standards, they will get around them. It is not claimed that the following guidelines will make programming in FORTRAN *easier* or *faster*. These guidelines are for the benefit of long-term maintenance or conversion of code. The STB plans to add tools to automate the use of these standards, but code reviews will be required to ensure their enforcement.

4.1 Documentation Standards

Existing local standards, as established in the mission planning and analysis domain at JSC, require each program unit to be documented in a prolog using standardized comment statements, referred to as CD statements. A program that meets these standards for documentation and the standards for the COMMON structure is called a COMGEN-compatible program. The original standards for the information contained in each of the types of CD statement are briefly documented in the appendix. Reengineering experience, which was gained in performing a pilot project to convert an existing FORTRAN program to Ada, has prompted additional standards related to documentation.

Following are new in-line documentation standards. The first refers to comments embedded in the body of the code; the remainder are comments to be provided in the prologs to subprograms:

- Header comment statements are to be embedded before blocks of code to improve the understandability of the function performed by the following code.

- By current standards, the CD1 statements in a subprogram's prolog provide a short description of the purpose of the subprogram. This description is to be expanded to include the identification of the requirement(s) that are met by the subprogram.
- By current standards, the CD7 statements in a subprogram's prolog record the functional description of the subprogram. By the new standard, this description is to be expanded to explain *how* and *why* of the design of the subprogram, in addition to *what* the subprogram does. It should not just be a restatement in English of the flow of the code. It should be written by someone familiar with the algorithm.
- If COMMON blocks are used, the variables must be documented either with the STB's tools or by using the CD4 statements of the prolog.
- New PACKAGE and VISIBLE fields to support virtual packages are to be included in the CD0 statements, which by current standards contain the identification information in a subprogram's prolog. These new fields are defined and explained in section 4.4

Figure 1 illustrates the structure of a COMGEN-compatible subprogram. Maintainability is improved by the in-line storage with the code of the following: documentation (standard CD documentation statements in subprogram prolog), requirements (CD1 statements), and design (CD7, CD8, CD9 statements and code block headers).

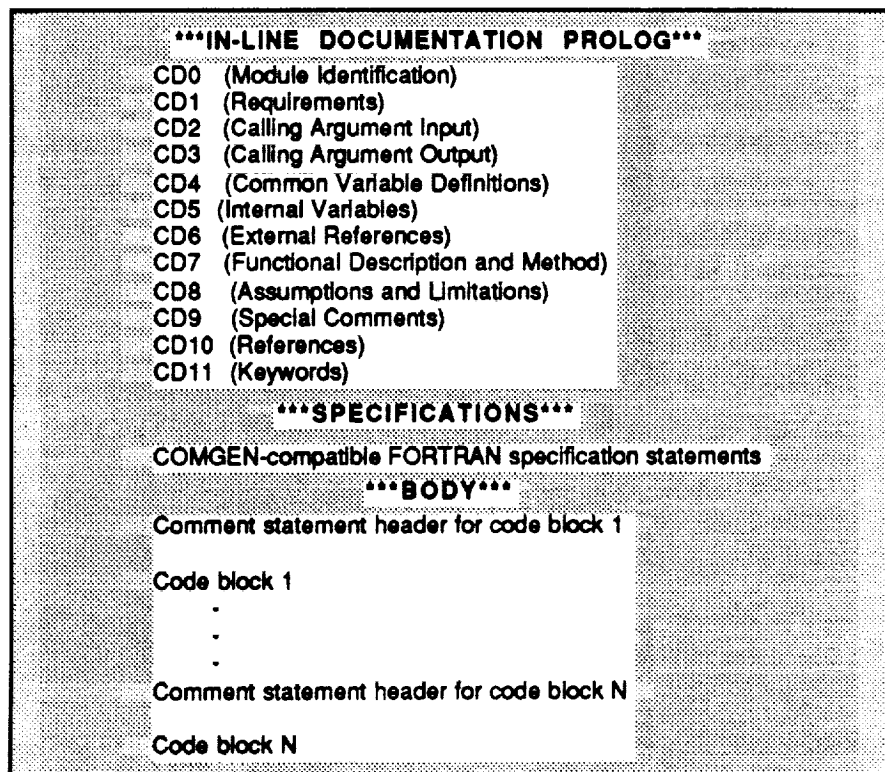


Figure 1. Structure of a COMGEN-Compatible Subprogram

Another type of documentation, which is often overlooked, is the names of source files. Under the new standards, source file names will match the subprogram names if the operating system allows it. Also, if allowed by the operating system, file name prefixes for owning packages is a useful convention. The VAX Ada conventions for the names of package specifications, package bodies, and separate units are good examples to follow.

4.2 Longer Variable Names

The use of longer, more meaningful variable names improve the readability of source code. For those worried about the possible need to return the code to the present six character format for portability to a compiler without an extension supporting longer names, one of the following two methods can be used:

- Use meaningful variable names with a maximum of 24 characters and if it becomes necessary later to convert them back to the six character format, perform global search-and-replace edits. The STB plans to provide tools to support the capability of reducing 24-character names to 6 characters, if it is needed.
- Leave the variables in six character format, but place comments in the CD9 statements, which contain special comments, in the following form:

CD9 short_name long_name

This will provide information that can be used later to change to the longer names when the code is moved to the FORTRAN 8x standard or is translated to another language.

4.3 Modern Control Flow Structures

As far as FORTRAN allows, use only the standard structured programming constructs. Reduce the use of the GO TO; use the GO TO only if no structured option is available. To accomplish these goals, the use of the FORTRAN 77 Block IF and the following constructs, which are supported by most compilers, are recommended:

- Block DO.
- DO WHILE.

Both of these constructs are military extensions to FORTRAN 77 and are in the draft FORTRAN 8x standards.

4.4 Grouping Subprograms into Virtual Packages

The Ada package and C file concepts are among the most useful features of these languages. These concepts group logically related subprograms together at a higher level of abstraction, the module level. In this document this module level of abstraction in FORTRAN programs is referred to as a "virtual package." Procedures and tools can provide the capability of defining virtual packages in FORTRAN 77. It requires a somewhat artificial technique, making use of

comment statements and specification data files to define the virtual packages. However, the use of these techniques will aid in code maintenance. In addition, it will allow the STB's automated documentation tools to produce even better documents than currently possible since it will properly group the separate subprograms into higher level modules.

To identify which package each subprogram is contained in, the addition of the following two fields to every FORTRAN subprogram's CD0 statements is needed:

- Statement of the form "PACKAGE = package_name".
- Statement of the form "VISIBLE = yes or no".

Each subprogram in a virtual package will have the same package_name field value. The specification/implementation separation of Ada can be provided by the setting VISIBLE = YES for externally visible subprograms and setting VISIBLE = NO for subprograms internal to the virtual package. VISIBLE = YES subprograms are called by subprograms with different values of PACKAGE. Subprograms with VISIBLE = NO are called only by subprograms with the same value of PACKAGE. Figure 2 compares the form of an Ada package and the form of a FORTRAN virtual package implemented through an external Module Definition Table (MDTAB) emulating the Ada Package specification with the package identification statements in the FORTRAN subprograms' in-line documentation prologs.

A pre-processing tool will be developed to help ensure compliance. However, a tool cannot guard against intentional misuse of the system. For instance, if the PACKAGE field were set to the same value for every subprogram in the program, or if the pre-processing tool were never run against the program, the procedure would not work. Code reviews are required to guard against this problem.

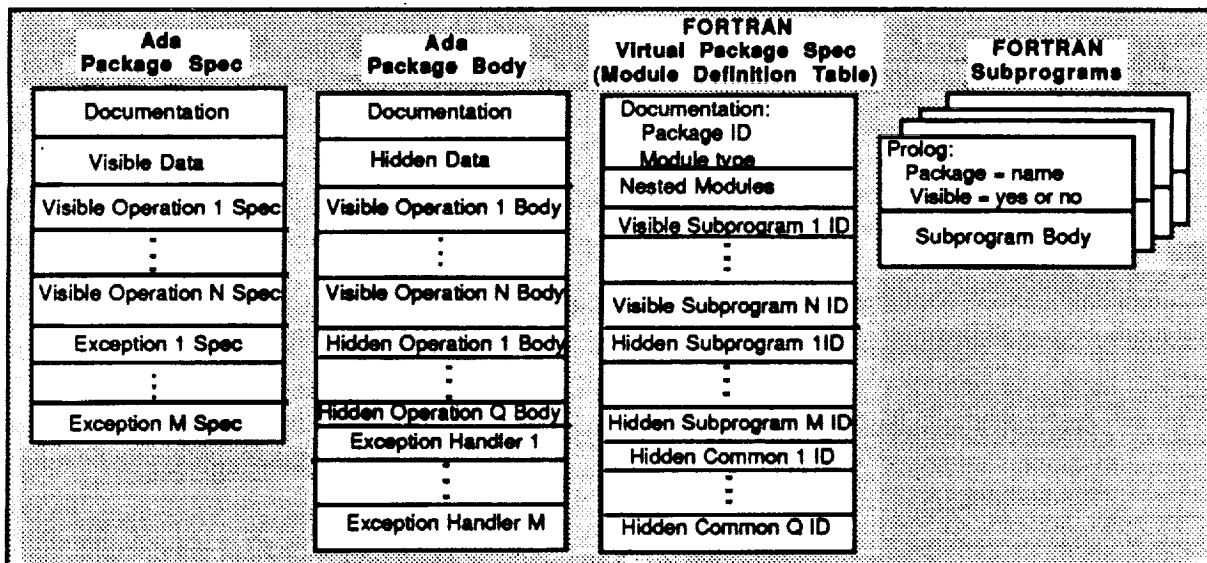


Figure 2. Comparison of Ada Package and FORTRAN Virtual Package

Based on the CD0 package identification statements, the program element table (PETAB) used by the STB's documentation tools will be modified to include the virtual package name for each subprogram. Also a new data set will be defined that will show the following for each virtual package:

- Names of the files that form the virtual package.
- Names of the visible entry points to the virtual package.
- List of the COMMON blocks to be "hidden" in the virtual package.

It is recommended that interfaces with virtual packages be clearly documented in interface control documents (ICD), and that these interfaces be managed by the development or maintenance team. Documentation and enforcement of stable interfaces improves maintainability and can help with problem identification.

4.5 Data Structuring

In building a FORTRAN program of any size, the logical blocks of the program are broken up into logical units and coded as subprograms. The primary method of inter-subprogram communication has traditionally been by the use of FORTRAN labeled COMMON blocks. The alternative method is by use of calling parameters.

4.5.1 Inter-Subprogram Communication via Calling Parameters

Program interfaces via COMMON blocks are much harder to maintain than interfaces via subprogram calling parameters. Although calling parameters can be less machine-efficient than passing values via COMMON blocks, in most cases the increased maintainability is worth the performance cost. It is preferable for virtual packages to communicate with other virtual packages via calling parameters. Calling parameters are preferable for internal communication *within* virtual packages as well, but exceptions are more easily justified for data that is common to an entire package on the "package-body" level.

However, it is not possible to dogmatically require *all* virtual packages to communicate with other virtual packages *only* via calling parameters because of limitations in FORTRAN 77. Communication with a large virtual package entirely by the use of calling parameters may require so many calling parameters that the number of parameters could become a problem, possibly outweighing the drawbacks of using COMMON blocks. This cannot be solved in FORTRAN until the program is converted to FORTRAN 8x, which will support record structures that reduce the number of calling parameters. Until then, COMMON blocks will probably have to be used.

Communication with a small virtual package usually can use only calling arguments, and it is to be strongly recommended that calling parameters be used in these cases. However, in scientific work passing mathematical constants in this way to low level subprograms presents a problem. It also should be noted that high level executive subprograms that drive a program may not fit well into packages and may also need access to most of the data in the program.

This also must be treated on a case-by-case basis. Each project will have to decide if certain global COMMONs are to be allowed or if calling arguments are to be used.

4.5.2 Inter-Subprogram Communication via COMMON Blocks

The use of COMMON blocks must be carefully controlled. Use them only to pass single entities, document them well, and do not allow them to be used as catchalls to pass things like unrelated flags. Code reviews should be used to closely scrutinize COMMON block usage. The programmer/analyst should be prepared to explain why the COMMON block was necessary and preferable. "Because it was easier" is not a good reason. "Because I can show that performance would be unacceptable without it" is a good reason.

Subprogram communication by use of FORTRAN COMMON blocks works well only if the COMMON is well structured and documented. The standards presented in the following paragraphs are an effort to support these conditions. The following two major methods for COMMON block management have been used within the mission planning and analysis community over the years:

- INCLUDE capability.
- Standard COMMON database (CDB) and the tools that support it.

Both of these methods have their strengths and weaknesses. The main problem with them relates to the need for unique variable names. The next FORTRAN standard with its longer variable names and record structures will solve this problem. The standards presented here support the continued use of both methods with the only changes being some additions to the INCLUDE method for the purpose of documentation.

INCLUDE Method

In this method, the specifications for a given COMMON block are defined in one place and then included in each subprogram that needs to access that COMMON block. This method is available on several computer systems now and will be in the next ANSI standard for FORTRAN 8x. The INCLUDE files should map well to record structures in the newer languages. The only requirements that these new local standards place on this method are as follows:

- INCLUDE file must contain information on the identification and purpose of the COMMON block in the CD0 and CD1 comment statements, respectively.
- COMMON variables must be defined in either of two ways:
 - As CD4 statements in the INCLUDE file.
 - Or in a CDB as required by the standard COMMON concept.

COMMON Database (CDB) Method

This method is based on a series of tools that automate the maintenance, documentation, or restructuring of FORTRAN COMMON blocks. These tools provide an automated means for the generation and automatic insertion of COMMON specification statements into the code of FORTRAN subprograms.

The CDB contains the information necessary to regenerate all the required specification statements in every subprogram for those COMMON blocks that the programmer has placed under standard COMMON control. To do this the CDB contains the following:

- Detailed structure of each COMMON block.
- For each variable, its name, type, dimension, and definition.
- List, by subprogram, of the COMMON variables required in each subprogram.

Various tools have been developed to automate the creation, comparison, display, update and the verification of the CDB. These tools will be modified to support longer variable names and the optional automatic conversion of the COMMON structure to the INCLUDE format. The tools will also be upgraded to generate the structure of the COMMON in the newer languages if the code must be translated to FORTRAN 8x, Ada, or C.

4.6 Input/Output

Input/output statements should be separated from the rest of the program where possible, i.e., they should be encapsulated in separate subprograms as much as possible. The fewer the number of compilation units that contain the input/output statements of a program, the easier it will be to convert the program to another language. This standard introduces the problem of how to move the data from the input/output subprogram(s) to and from the low-level virtual packages that use and/or compute the data. There are three basic methods for accomplishing the data transfer:

- COMMON blocks.
- Calling argument parameters.
- Entry points in the package that send data to and receive data from the input/output subprograms via calling arguments.

Each of these methods has advantages and disadvantages. The choice of which to use will have to be made on a case by case basis.

5.0 SUMMARY

The standards presented in this volume are added to the existing standards defined in the mission planning and analysis domain at JSC. The goal of this set of standards is to improve maintainability and to permit relatively automated translations to newer languages. Where

FORTRAN Standards

FORTRAN 77 does not provide constructs, virtual constructs have been devised. The adoption of these standards at this time will give the JSC software community an early start on moving to the FORTRAN 8x standards which will include many of these capabilities. Table 1 summarizes the new FORTRAN coding standards and lists the primary benefit of each.

Table 1. Standards Summary

STANDARD	BENEFIT
Documentation Header statement before code blocks Requirements in CD1 statements Rationale in CD7 statements Virtual package identification	Understandability Understandability and traceability Design knowledge capture Maintenance
Longer, more meaningful variable names	Understandability
Modern control flow structures Block DO DO WHILE	Maintenance and understandability
Grouping subprograms into virtual packages	Higher level of abstraction, understandability
Data structuring Preferred use of calling parameters Controlled use of COMMON blocks INCLUDE COMMON database concept	Maintenance Maintenance
Preferably encapsulate input/output in separate subprograms	Maintenance and support to future conversions

6.0 APPENDIX

The appendix contains brief definitions of the standardized comment statements and descriptions of the tools in the STB tool set. For further information, the reader is referred to *Automated Software Documentation Techniques* (NASA) or *Computer Program Development and Maintenance Techniques* (NASA IN 80-FM-55).

6.1 Standardized Comment Statements (CDs)

The following is an annotated list of the standardized comment statements (known as CDs) that provide in-line documentation in a COMGEN-compatible subprogram.

CD0 IDENTIFICATION.

- Information on programmer, modifier, documenter, formulator:
- Name, organization, date.

CD1 PURPOSE.

- Short description of what the subprogram does, not how it does it.
- Longer description can follow, but short summary must be first.
- Topic sentence describes the module; the rest elaborates the module.
- May be the actual requirement paragraph(s) allocated to this module.

CD2 CALLING ARGUMENT INPUT.

- Name, dimension, type, length, definition.
- Optional extension for other inputs such as data files.

CD3 CALLING ARGUMENT OUTPUT.

- Name, dimension, type, length, definition.
- Optional extension for other outputs such as data files.

CD4 COMMON VARIABLE DEFINITIONS.

- Can be inserted automatically by the INSDOC processor (see following annotated list of STB tools) from definitions in the COMMON database (CDB).
- Optional extension for other global data such as data files.

CD5 INTERNAL VARIABLES.

- Name, dimension, type, length, definition.
- Optional extension for other local data such as temporary data files.

- CD6 EXTERNAL REFERENCES.
- External data files, external subprograms referenced, subprogram referenced by.
 - Normally not used – can be obtained from RELREF processor (see following annotated list of STB tools) or display of the ERTAB, the Elements Referenced Table.
- CD7 FUNCTIONAL DESCRIPTION AND METHOD.
- Logical flow of subprogram in narrative form.
 - Optionally includes PDL.
- CD8 ASSUMPTIONS AND LIMITATIONS.
- CD9 SPECIAL COMMENTS.
- CD10 REFERENCES.
- References to formal external documentation.
 - References to other related subprograms (i.e., SEE ALSO).
- CD11 KEYWORDS.

6.2 Tool Set

The following is an annotated list of the existing STB tool set that supports design recovery and reengineering.

- AUTODOC Automatic Subprogram Documentation Processor; tool that generates documentation reports using CD statements and data sets.
- CAUDIT Code Auditor; tool that checks for coding standards violations
- CCREF COMMON Block Cross-Reference Processor; tool that generates COMMON cross references.
-
- CLEANUP Cleanup Processor; tool that cleans up COMMON; inserts CD statements; alphabetizes specification statements; resequences statement numbers.
- COMPARE Symbolic File and Element Comparison Processor; tool that compares two files (recognizes some data sets and does smart comparisons).
- CONVERT Conversion Processor; tool that performs conversion operations on source code.
- CREATE Database Creation processor; tool that creates data sets.

FORTRAN Standards

DDT	Detailed Debug Trace Program; tool that inserts debug statements.
DEFINE	Documentation Definition Processor; tool that inserts definitions in CD statement section in free form.
DEPCHT	Dependency Chart Generator; tool that generates hierarchical call graphs.
DISPLAY	Data Set Display Processor; tool that displays a data set.
DOCGEN	Document Editor; tool that generates a document (ASCII source file processing).
DSDGEN	Data Structure Definition Processor; tool that allows a user to preview a data structure for use in DOCGEN or to build CD and/or V statements.
FORREF	FORTRAN Cross-Reference Display Processor; tool that generates symbol and statement number cross references.
INSDOC	COMMON Variable and Keyword Documentation Insertion Processor; tool that inserts CD4 variable definitions (with C option) and keyword CD11 statements (K option).
MAZE	Memory Map Analyzer; tool that displays memory map (uses Unix 'nm').
MERGE	File Merge Processor; tool that merges two data sets.
OMNIBUS	Omnibus Element Processor; tool that creates omnibus elements for symbol data sets.
QUERY	Query Processor; tool that queries data sets.
RELREF	Relocatable Element Cross-Reference Program; tool that generates calls/called-by list, including COMMON.
SCANPF	Control Script Generator; tool that generates control scripts.
SETGEN	Dependent Element Set Generator; tool that generates list of subprograms called in and below a given subprogram.
SPECNP	Common Variable Specification Statement Generation Processor; tool that restructures COMMON; generate COMMON specifications.
SUBDOC	Subprogram Documentation Processor; tool that generates subprogram descriptions using CD statements.
TABLES	Tables Processor; tool that generates tables for documents.

FORTRAN Standards

TOCGEN	Table of Contents (TOC) Generator; tool that displays table of contents for Unix directory, optionally includes program size information.
UPDATE	Database Update Processor; tool that updates a data set.
VERIFY	Database Verification Processor; tool that verifies a data set.

7.0 GLOSSARY

Acronyms:

CD	CD statement, the in-line documentation standard in the mission planning and analysis domain. For further information, the reader is referred to <i>Computer Program Development and Maintenance Techniques</i> , NASA IN 80-FM-55.
CDB	COMMON database of a program meeting the standard COMMON concept in the mission planning and analysis domain; primary documentation of a FORTRAN program's COMMON structure; used by many of the STB's tools. For further information, the reader is referred to <i>Computer Program Development and Maintenance Techniques</i> , NASA IN 80-FM-55.
FADS	Flight Analysis and Design System.
ICD	Interface Control Document.
JSC	Johnson Space Center.
MOD	Mission Operations Directorate.
PETAB	Program Element Table.
STB	Software Technology Branch.

Terms:

arbitrary FORTRAN	FORTRAN program that is not compatible with the COMGEN standards in place for JSC's mission planning and analysis domain.
COMGEN-compatible	FORTRAN program that is compatible with the COMGEN standards in place for JSC's mission planning and analysis domain. For further information, the reader is referred to <i>Computer Program Development and Maintenance Techniques</i> , NASA IN 80-FM-55.
COMMON database	COMMON database of a program meeting the standard COMMON concept in the mission planning and analysis domain; primary documentation of a FORTRAN program's COMMON structure; commonly referred to as the CDB; used by many of the STB's tools. For further information, the reader is referred to <i>Computer Program Development and Maintenance Techniques</i> , NASA IN 80-FM-55.
design recovery	Reverse engineering, the first step for maintenance or reengineering.

FORTTRAN Standards

environment	Instantiation of a framework, i.e., an integrated collection of tools. It may support one or more methodologies and may also provide a framework for third party tools.
framework	Software system to integrate both the data and the control of new and existing tools; usual components include a user interface, object management system, and a tool set.
FORTTRAN 77	ANSI standards for FORTTRAN in effect in 1990.
FORTTRAN 8x	Future ANSI standards for FORTTRAN; expected to be approved and released soon; draft standards have been circulated; unofficially referred to as FORTTRAN 90.
forward engineering	Process of developing software from "scratch," through the phases of requirements, design, and coding.
package	"A collection of logically related entities or computational resources" (Booch).
reengineering	"The examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form" (Chikofsky and Cross, 1990); combination of reverse engineering and forward engineering.
reverse engineering	"The process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction" (Chikofsky and Cross, 1990); the first step for maintenance or reengineering; reverse of forward engineering; process of starting with existing code and going backward through the software development life cycle.
software maintenance	"Process of modifying existing operational software while leaving its primary functions intact" (Boehm, 1981).
subject program	Program that is being maintained or reengineered.
subprogram	Subroutine or function.
"X"-option	Reference to a mechanism of using a letter on the execute statement to indicate the control path to be followed in the execution of a program or processor; used by the Unisys, Unix, and DOS operating systems.
virtual package	Package concept as defined by Booch, but implemented either in Ada, which enforces the concept, or in a language in which the concept must be supported procedurally.

8.0 REFERENCES

Boehm, B. W., *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

Booch, Grady, *Software Engineering with Ada*, Menlo Park, CA: Benjamin/Cummings Publishing Co., Inc., 1983.

Booch, Grady, *Software Components with Ada*, Menlo Park, CA: Benjamin/Cummings Publishing Co., Inc., 1987.

Brale, Dennis, *Computer Program Development and Maintenance Techniques*, NASA IN 80-FM-55, Houston, TX: NASA Johnson Space Center, November 1980.

Brale, Dennis, *Automated Software Documentation Techniques*, Houston, TX: NASA Johnson Space Center, April 1986.

Brale, Dennis, *Software Development and Maintenance Aids Catalog*, NASA IN 86-FM-27, Houston, TX: NASA Johnson Space Center, October 1986.

Brale, Dennis, "A Software Recovery Methodology," unpublished internal document, Houston, TX: NASA Johnson Space Center, FR51, January 1990.

Brale, Dennis, "FORTRAN Standards for Future Translation and/or Design Recovery," unpublished internal document, Houston, TX: NASA Johnson Space Center, FR51, January 1990.

Brale, Dennis and Allan Plumb, "An Environment for Software Conversion and Code Recovery," unpublished internal document, Houston, TX: NASA Johnson Space Center, FR51, March 1990.

Brale, Dennis and Allan Plumb, *Maintenance Strategies for Design Recovery and Reengineering: Methods*, Volume 3, Houston, TX: NASA Johnson Space Center, June 1990.

Brale, Dennis and Allan Plumb, *Maintenance Strategies for Design Recovery and Reengineering: Concepts for an Environment*, Volume 4, Houston, TX: NASA Johnson Space Center, June 1990.

Chikofsky, Elliot J. and James H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, January 1990.

Clark, Robert G., *Programming in Ada: A First Course*, Cambridge: Cambridge University Press, 1985.

Fridge, Ernest M., *Maintenance Strategies for Design Recovery and Reengineering: Executive Summary and Problem Statement*, Volume 1, Houston, TX: NASA Johnson Space Center, June 1990.

FORTRAN Standards

George, Vivian and Allan Plumb, *A Method for Conversion of FORTRAN Programs*,
Houston, TX: Barrios Technology, Inc., January 1990.